VERA C. RUBIN OBSERVATORY

Throwing coins into a VO:TNG wishing well

Frossie Economou, Russ Allbery, Gregory Dubois-Felsmann

Rubin Observatory

NSF AURA U.S. DEPARTMENT OF ENERGY SLAC CHARLES AND LISA SIMONYI FUND FOR ARTS AND SCIENCES LSST CORPORATION

Storytime…

The uncanny valley of VO web services

- Current web frameworks (we use FastAPI) implement web APIs with strong RESTful semantics
- Their popularity in the tech sector has given rise to a rich ecosystem of developer infrastructure and automation that greatly boosts velocity and know-how that is easily transferable between projects
- These frameworks also offer hardening against security exploits
- API Bearer tokens are ubiquitous
- Python has won
- Cloud architectures are increasingly de-facto even in astronomy
- XML vs JSON is the most visible part of this mismatch (and it's an important one) but this is an issue that goes beyond serialization format

Example 1: Case (in)sensitivity

- DALI requires case insensitivity
- This is highly unusual and makes implementing VO standards in common web frameworks pointlessly hard
- For example, FastAPI would be able to automatically impose restrictions on allowable values for parameters and automatically generate error messages if its normal query and form parameter parsing could be used; but like other python frameworks, it treats parameter names as case-sensitive
- Doing this work ourselves instead of leveraging the same tooling we use in non-VO web services was tedious ditch-digging that resulted in code that is harder to maintain and reduced the accuracy of OpenAPI docs

Example 2: GET/POST and the security model

- DALI requires synchronous resources be available via both `GET` and `POST`.
- Implementing any resource that creates state changes in the server via `GET` violates the HTTP security model and thus makes the service more vulnerable to CSRF attacks.
- Requiring `GET` be supported means the only options for an implementation concerned about this risk is to not support sync requests or not follow the standard.ee, for example, the OWASP CSRF Prevention Cheat Sheet: “Do not use GET requests for state changing operations.”
- Other security-weak examples include http(!s) XML schema references, no provision for clients to get a form token for POST requests, etc.

Example 3: And yes, XML

- XML is not well supported in current web frameworks
- Generating and parsing XML is tedious
- Parsing XML is prone to a plethora of security issues
- We literally don’t use XML anywhere else
- Whereas in modern web frameworks, JSON generation and parsing is built-in and transparent

We gotta keep up

- We increasingly depend on non-astronomy back-end developers
- Aggressive data sizes and new compute models are driving us into areas where deployment, security and scalability engineering is a requirement (cf Rubin TAP queries can exhaust any resources you have on your TAP server)
- To create a vibrant developer community in the next generation, we should consider “normalising” the experience of developing VO web services to match that of non-VO web services as much as possible.
- We are aware this proposal is a “breaking” change, and as a community we should discuss the evolution vs revolution pros and cons (VO 2.0?)
- This is not the first time this will have to happen either; we should consider what processes and patterns would allow us to keep up with evolving technology in this area. The issue is not that we have bad standards, but that being divergent from current frameworks is tedious, error-prone and confers no benefits.

More examples at https://sqr-063.lsst.io